Oracle® OLAP

Developing Analytic Workspace Manager Plug-ins

11*g* Release 1 (11.1)

E12377-03

September 2009

This technical note describes how to create a Java plug-in for use with Oracle Analytic Workspace Manager, Version 11.1. With a plug-in, a Java developer can extend the functionality of Analytic Workspace Manager.

This technical note includes the following topics:

- Introducing Analytic Workspace Manager Plug-ins
- Analytic Workspace Manager Plug-in Interface and Example

See Also:

- *Oracle OLAP User's Guide*
- *Oracle OLAP Java API Reference*
- *Oracle OLAP Java API Developer's Guide*

Introducing Analytic Workspace Manager Plug-ins

An Analytic Workspace Manager plug-in enables you to run Java code in the context of Analytic Workspace Manager. With a plug-in, you can implement user interfaces for programs that perform actions such as the following:

- Create new types of calculations
- Create forecasts
- Create custom OLAP metadata objects, such as an enterprise-specific time dimension

In an Analytic Workspace Manager plug-in, you can use the following Java APIs:

- Oracle OLAP Java API
- JDBC API
- Swing API

You can invoke OLAP DML or SQL procedures by using JDBC classes.

Enabling Analytic Workspace Manager Plug-ins

Analytic Workspace Manager has a configuration option that specifies whether or not it uses plug-ins. To enable plug-ins, from the Analytic Workspace Manager **Tools** menu, select **Configuration**, as shown in Figure 1. In the **Configuration** dialog box, select **Enable Plugins** and specify the directory that contains your plug-ins, as shown in Figure 2. Click **OK** and then exit and restart Analytic Workspace Manager.

Figure 1 Configuration Item on the Tools Menu

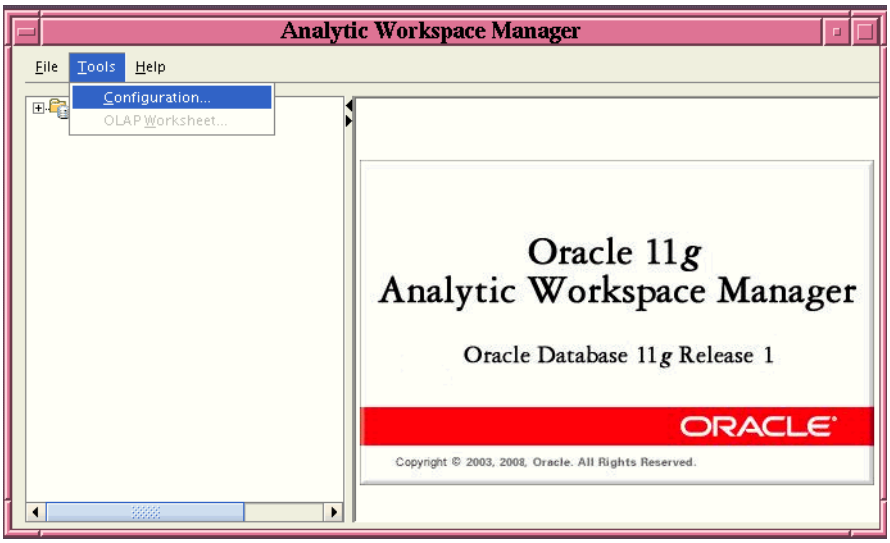
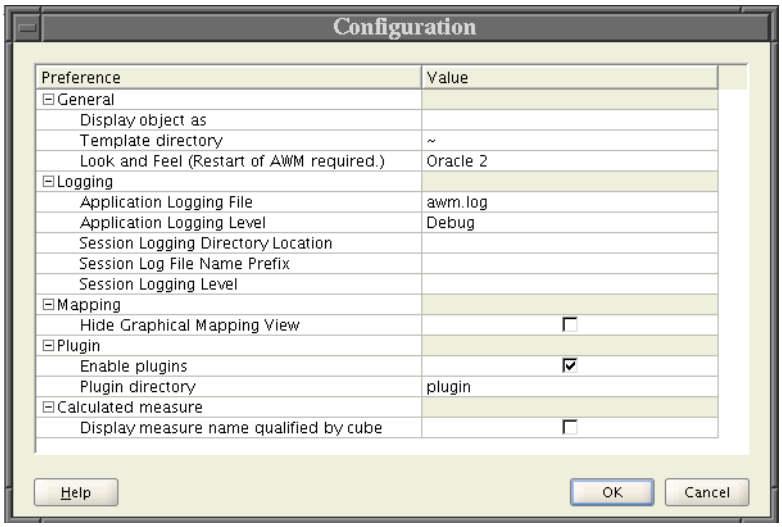


Figure 2 Configuration Dialog Box with Enable Plugins Selected



How Analytic Workspace Manager Calls a Plug-in

If Analytic Workspace Manager has plug-ins enabled, then on startup Analytic Workspace Manager dynamically loads Java code from JAR files located in the plug-ins directory. After loading the contents of the JAR files, Analytic Workspace Manager looks for classes that implement the `AWMPlugin` interface.

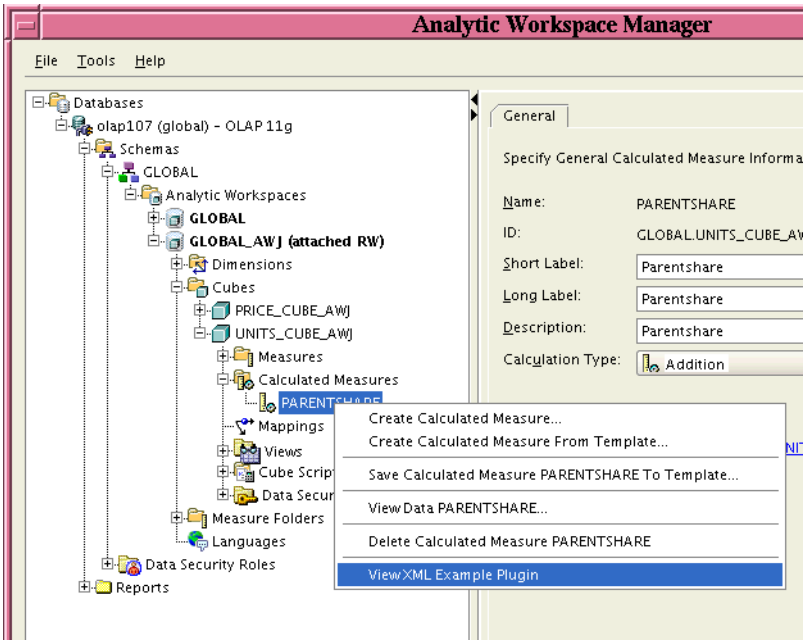
Note: You can include multiple plug-ins in a single JAR file.

When a user right-clicks an object in the Analytic Workspace Manager navigation tree, a menu appears that presents the actions available for the object. The menu also

displays the plug-ins that apply to the object. A plug-in uses the `isSupported` method to indicate whether it applies to an object in the tree.

The menu displays the text returned by the `getMenu` method of the plug-in. Figure 3 shows the menu that Analytic Workspace Manager displays for a right-click on a calculated measure in the tree. The menu includes the `ViewXMLPluginExample` example plug-in. For the code of the plug-in example, see Example 2.

Figure 3 Right-click Menu of the Navigation Tree



If the user selects the plug-in, then Analytic Workspace Manager calls the `handle` method of the plug-in. The `handle` method specifies the actions that the plug-in performs. The `refreshTree` method of the plug-in indicates whether Analytic Workspace Manager refreshes the navigation tree to include any new objects created by the plug-in.

Analytic Workspace Manager passes the following objects as input parameters to the plug-in methods:

- For the `conn` parameter, a `java.sql.Connection` object that represents the current connection to the Oracle Database instance.
- As of the 11.1.0.7 release of Oracle Database, for the `type` parameter, a `java.lang.String` object that indicates a type designation that Analytic Workspace Manager assigns to the object. For a description of type parameter values, see "Values for the type and object Parameters". Before 11.1.0.7, the type parameter was only used internally by Analytic Workspace Manager.
- For the `obj` parameter, a `java.lang.Object` that Analytic Workspace Manager associates with the object selected in the Analytic Workspace Manager navigation tree. The `Object` can be a `String`, an object from the Oracle OLAP Java API, or `null`.
- For the `aw` parameter, Analytic Workspace Manager passes in `null`. This parameter exists for compatibility with 10*g* plug-ins.

- For the `params` parameter, a `java.util.Map` object that contains the following key/object pairs:
  - The `String` `DATAPROVIDER` is the key for an `oracle.olapi.metadata.mdm.MdmMetadataProvider`.
  - The `String` `DATASOURCE` is the key for a `java.sql.DataSource`.

For the `handle` method, Analytic Workspace Manager also passes a `java.awt.Frame` object that the plug-in can use as the parent frame for user interface components.

Analytic Workspace Manager does not pass any user identification or password to the plug-in. It only passes the connection object. An Analytic Workspace Manager plug-in does not allow you to do anything that you cannot do by writing a standalone Java program.

Steps in Creating a Plug-in

The prerequisites for creating an Analytic Workspace Manager plug-in are the following:

- Include the `AWMPlugin` interface class in your development environment. You can download a JAR file that contains the `AWMPlugin.class` file from the Oracle Technology Network (OTN) Web site for Oracle OLAP at `http://www.oracle.com/technology/products/bi/olap/index.html`. You can also download the `AWMPlugin.java` file itself from that site.
- Include the `awxml.jar` and `olap_api.jar` files in your development environment. The `olap_api.jar` contains the classes in the Oracle OLAP Java API. The `awxml.jar` contains the `class` file for the `oracle.AWXML.AW` class, which the `AWMPlugin` interface includes for compatibility with the 10*g* release of Analytic Workspace Manager. These JAR files are located in the `/olap/api/lib` directory under the *ORACLE_HOME* directory in an Oracle Database installation.
- Compile the code with JDK 1.5.

> **Note:** Only plug-ins compiled with JDK 1.5 are compatible with Analytic Workspace Manager in 11*g* Release 1 (11.1).

To create an Analytic Workspace Manager plug-in, do the following:

1. Create a class that implements the `AWMPlugin` interface.
   - In the `isSupported` method, specify the objects in the navigation tree to which the plug-in applies.
   - Have the `getMenu` method return the text to display on the right-click menu for navigation tree objects that the plug-in supports.
   - In the `handle` method, include the code for the operations that the plug-in performs.
   - Have the refreshTree method return a boolean that specifies whether or not to refresh the navigation tree.
2. Using JDK 1.5, compile the plug-in and any other classes that it uses.
3. Deploy the plug-in and other classes to a JAR file. You can include more than one plug-in in the same JAR file.

4. Put the JAR file in the plug-ins directory.
5. Start Analytic Workspace Manager.

> **Note:** Analytic Workspace Manager only loads the contents of the JAR files upon startup, so if you put a new or updated version of a JAR file in the plug-ins directory, then you must restart Analytic Workspace Manager.

Values for the type and object Parameters

For the `type` parameter of the methods of an `AWMPlugin` implementation, Analytic Workspace Manager passes to the plug-in a label that identifies the type of the navigation tree object for which the plug-in is invoked. For the `obj` parameter of the methods, Analytic Workspace Manager passes an `Object`, which is a `java.lang.String` or an OLAP metadata object, or `null`.

A plug-in can use the `type` value to distinguish between the navigation tree objects that are associated with the same metadata object. For example, for all of the top-level navigation tree objects under a dimension, Analytic Workspace Manager passes as the `obj` parameter the same `MdmPrimaryDimension` object, but it passes a different `type` label for each navigation tree object.

Table 1 shows the `type` parameter values and `obj` parameter objects that Analytic Workspace Manager passes to the plug-in for the selected navigation tree object. The indentation of objects in the Navigation Tree Object column indicates the hierarchy of the tree. Text in italics indicates a variable object name. The `AW` object is an `oracle.olapi.metadata.deployment.AW` object. The other metadata objects, such as `MdmStandardDimension` and `MdmCube`, are classes in the `oracle.olapi.metadata.mdm` package. The Reports object and all of the objects under it have the same type.

Table 1 Type Values and Objects for Navigation Tree Objects

Navigation Tree Object	type Parameter Value	obj Parameter Object
Databases	`Databases`	`null`
Database name	`DATABASE`	`String` *Database identifier*
Schemas	`SCHEMA_FOLDER`	`String` *Database identifier*
Schema name	`SCHEMA`	`String` *Schema name*
Analytic Workspaces	`WORKSPACE_FOLDER`	`String` *Schema name*
Analytic workspace name	`WORKSPACE`	`AW`
Dimensions	`DIMENSION_FOLDER`	`AW`
Dimension name	`DIMENSION`	`MdmStandardDimension` or `MdmTimeDimension`
Levels	`DIMENSION_LEVEL_FOLDER`	`MdmStandardDimension` or `MdmTimeDimension`
Level name	`DIMENSION_LEVEL`	`MdmDimensionLevel`
Hierarchies	`DIMENSION_HIERARCHY_FOLDER`	`MdmStandardDimension` or `MdmTimeDimension`
Hierarchy name	`DIMENSION_HIERARCHY`	`MdmLevelHierarchy` or `MdmValueHierarchy`
Attributes	`DIMENSION_ATTRIBUTE_FOLDER`	`MdmStandardDimension` or `MdmTimeDimension`

Table 1 (Cont.) Type Values and Objects for Navigation Tree Objects

Navigation Tree Object	type Parameter Value	obj Parameter Object
Attribute name	DIMENSION_ATTRIBUTE	MdmBaseAttribute
Mappings	DIMENSION_MAP	MdmStandardDimension or MdmTimeDimension
Views	DIMENSION_VIEW_FOLDER	MdmStandardDimension or MdmTimeDimension
View name	DIMENSION_VIEW	MdmStandardDimension or MdmTimeDimension
Data Security	DATA_SECURITY	MdmStandardDimension or MdmTimeDimension
Cubes	CUBE_FOLDER	AW
Cube name	CUBE	MdmCube
Measures	CUBE_MEASURE_FOLDER	MdmCube
Measure name	CUBE_MEASURE	MdmBaseMeasure
Calculated Measures	CUBE_DERIVED_MEASURE_FOLDER	MdmCube
Calculated measure name	CUBE_DERIVED_MEASURE	MdmDerivedMeasure
Mappings	CUBE_MAP	MdmCube
Views	CUBE_VIEW_FOLDER	MdmCube
View name	CUBE_VIEW	MdmCube
Cube Scripts	null	MdmCube
Cube script name	null	null
Data Security	DATA_SECURITY	MdmCube
Measure Folders	MEASURE_FOLDER_FOLDERS	AW
Measure folder name	null	MdmOrganizationalSchema
Languages	LANGUAGE	AW
Data Security Roles	ACL_DOCUMENT_FOLDER	null
Reports	AWMTREE_REPORT	null

Analytic Workspace Manager Plug-in Interface and Example

This section contains the specification for the `AWMPlugin` interface and an example that implements the interface.

Plug-in Interface Specification

Example 1 has the code for the interface. The example leaves out the documentation comments that appear in the `AWMPlugin.java` file that is available on the Oracle OLAP OTN Web site. The methods and their input parameters are described in "How Analytic Workspace Manager Calls a Plug-in".

Example 1 The AWMPlugin Interface

```
package oracle.olap.awm.t;

import oracle.AWXML.AW;
import java.awt.Frame;
import java.sql.Connection;
import java.util.Map;
```

```
public interface AWMPlugin
{
  boolean isSupported(Connection conn, String type, Object obj, AW aw,
                      Map params);

  String getMenu(Connection conn, String type, Object obj, AW aw,
                 Map params);

  void handle(Frame parent, Connection conn, String type, Object obj,
              AW aw, Map params);

  boolean refreshTree(Connection conn, String type, Object obj, AW aw,
                      Map params);
}
```

Example of an Analytic Workspace Manager Plug-in

Example 2 contains the code for the `ViewXMLPluginExample` class, which implements the `AWMPlugin` interface. The plug-in applies to `oracle.olap.metadata.mdm.MdmBaseMeasure` and `oracle.olap.metadata.mdm.MdmDerivedMeasure` objects, which correspond to the Measure and Calculated Measure objects, respectively, in the Analytic Workspace Manager navigation tree. The plug-in gets and displays an XML representation of the measure. For an example of the message that `ViewXMLPluginExample` displays, see Figure 4.

The example does not include the documentation comments of the methods of the `AWMPlugin` interface or the input parameters and return values of those methods. The documentation comments appear in the `ViewXMLPluginExample.java` file that is available on the Oracle OLAP OTN Web site.

Example 2 The ViewXMLPluginExample Class

```
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.sql.Connection;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import oracle.AWXML.AW;
import oracle.olap.awm.plugin.AWMPlugin;
import oracle.olapi.metadata.mdm.MdmBaseMeasure;
import oracle.olapi.metadata.mdm.MdmDerivedMeasure;
import oracle.olapi.metadata.mdm.MdmMetadataProvider;
import oracle.olapi.metadata.mdm.MdmObject;
```

```
/**
 * An implementation of the AWMPlugin interface that displays the XML
 * representation of an Oracle OLAP measure object.
 */
public class ViewXMLPluginExample implements AWMPlugin
{
  public boolean isSupported(Connection conn, String type, Object obj,
                             AW aw, Map params)
  {
    // Support MdmBaseMeasure and MdmDerivedMeasure objects.
    if (obj instanceof MdmBaseMeasure || obj instanceof MdmDerivedMeasure)
    {
      return true;
    }
    return false;
  }

  public String getMenu(Connection conn, String type, Object obj, AW aw,
                        Map params)
  {
    // Text to display on the right-click menu.
    String menu = "View XML Example Plug-in";
    return menu;
  }

  public boolean refreshTree(Connection conn, String type, Object obj, AW aw,
                             Map params)
  {
    // This example does not create new metadata objects, so return false.
    return false;
  }

  public void handle(Frame parent, Connection conn, String type, Object obj,
                     AW aw, Map params)
  {
    if (obj instanceof MdmObject)
    {
      // Get the MdmMetadataProvider to use in exporting the XML.
      Object objdp = params.get("DATAPROVIDER");
      if (objdp != null)
      {
        MdmObject mobj = (MdmObject)obj;
        MdmMetadataProvider mdp = (MdmMetadataProvider)objdp;

        // Get the XML representation of the MdmObject.
        List objects = new ArrayList();
        objects.add(mobj);
        Map renameMap = null;
        boolean includeOwnerString = true;
        String title = "XML for " + mobj.getName();
        try
        {
          String xml =
            mdp.exportFullXML(objects, renameMap, includeOwnerString);
          // Create a dialog box and display the XML.
          DisplayXMLDialog dxd = new DisplayXMLDialog(parent, title, true,
                                                      xml);
          dxd.setVisible(true);
        }
```

```
      catch (IOException ie)
      {
        //Ignore error.
      }
    }
  }

/**
 * An inner class that creates a dialog box that displays the XML.
 */
class DisplayXMLDialog extends JDialog implements ActionListener
{
  /**
   * Creates a DisplayXMLDialog for displaying the contents of the xml
   * parameter.
   *
   * @param parent A Frame that is provided by Analytic Workspace Manager.
   * @param title A String that contains text to use as the title for the
   *              dialog box.
   * @param modal A boolean that specifies whether or not the dialog box is
   *              modal.
   * @param xml A String that contains the XML to display.
   */
  public DisplayXMLDialog(Frame parent, String title, boolean modal,
                          String xml)
  {
    setLocation(200, 200);
    setTitle(title);
    setModal(modal);

    try
    {
      displayXML(xml);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  /**
   * Creates a dialog box and displays the contents of a String.
   *
   * @param xml A String that contains the XML to display.
   */
  private void displayXML(String xml)
  {
    JTextArea ta = new JTextArea(xml);
    ta.setEditable(false);
    Font of = ta.getFont();
    Font f = new Font("Courier New", of.getStyle(), of.getSize());
    ta.setFont(f);

    JScrollPane p = new JScrollPane();
    p.getViewport().add(ta);

    JPanel buttonPane = new JPanel();
    JButton button = new JButton("Close");
    buttonPane.add(button);
```

```
      button.addActionListener(this);
      getContentPane().add(buttonPane, BorderLayout.SOUTH);

      getContentPane().add(p, BorderLayout.NORTH);
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
      pack();
      setVisible(true);
    }

    /**
     * Performs an action for the Close button.
     *
     * @param e An ActionEvent for the Close button.
     */
    public void actionPerformed(ActionEvent e)
    {
      setVisible(false);
      dispose();
    }
  }
}
```

Figure 4 illustrates the type of dialog box that `ViewXMLPluginExample` displays.

Figure 4 Dialog Box Displayed by the Example Plugin

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Developing Analytic Workspace Manager Plug-ins, 11*g* Release 1 (11.1)
E12377-03